Installing Linux: Post-Install

Fall 2020 - Ben Cuan
Slides adopted from Ethan Smith

Administrivia

- **Lab 3 due Thursday, 10/8 11:59pm**
  - In order to pass, you can only have up to 2 late labs. Email us at decal@ocf.berkeley.edu if you didn't complete the first 2 labs so we can make sure you pass!

- **Come to the live session this Thursday, 10/8 at 8pm!!**
  - I'll be giving a walkthrough on configuring users on a fresh machine
  - Will also be talking a bit about GUI (Xserver) configs and ricing. Come if you want to learn how to make your linux install work and look sleek af

- Lab 4 released and is due next Thursday.
  - This one's very open ended choose your own adventure, so have fun with it!

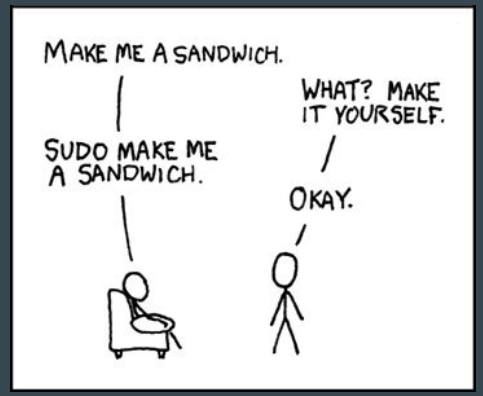
The system is installed: now what?

We're not done yet

We need to:

- Make users and groups
- Assign and drop privileges
- Secure the system on the network
- Install basic utilities
- Make this a usable system

Users and Groups



Users and Groups

- *NIX access control based on user/group paradigm - ACLs
- Regular User/Groups -> filesystem and execution permissions
  - Files are owned by users/groups, and opened/executed with the privileges of the executing program
  - A virtual terminal is opened by /bin/login (exec'd as root, since it was started by /sbin/init), then changes users to you and launches your shell (e.g. /bin/zsh) as you, and then your shell is the parent process of any programs you start

Why is this important?

Users and Groups from a Security Standpoint

- The kernel controls access to files and processes based on **uid**s and **gid**s
  - the kernel raises EACCES (“Permission Denied”) system error if you don’t have the right one
- These checks occur in privileged kernel memory - the CPU itself disallows memory accesses to other memory when not in kernel mode, by using protection rings.
  - Ring 3: user mode, Ring 0: supervisor mode
- The root user (uid/gid 0) has all permissions in the ‘classical’ UNIX permissions system - the root user can execute arbitrary programs, read any file, etc.
  - If a program running as uid 0 gets exploited, what does this mean?

Almost all hardware/kernel protections are negated if the root user is compromised

Making Users

- Users: call `useradd` or friendly `adduser`
  - Options: create home directory, set shell, add to groups, set expire time, etc.
  - Adds an entry to /etc/passwd containing GECOS information
- Use `passwd` to set passwords - adds hashed password to /etc/shadow

Managing Users

- List users by `cat /etc/passwd`, `getent passwd`, etc.
- Delete users by doing `userdel`
  - Be wary of deleting users - file ownership is governed by uid, and newly created users claim the next available UID. Make sure to fix file permissions when doing this.
- System users: UIDs in the SYS_UID range (100-499), usually for daemons and such things (e.g. www-data, nobody)

Making and Managing Groups

- Users by default have a primary group - usually named after their username
- Can be added to secondary groups - e.g. `wheel`, `sys`, etc.
- Add new group by doing “groupadd”, remove by doing “groupdel”
  - Friendlier equivalents are `addgroup` and `delgroup` on Debian
- List people in groups by doing `groups`, `getent group <group>`, cat `/etc/group`, etc.

Switching User - su

- su = switch/substitute user
- sg = switch/substitute group
- Useful if you want to modify something for a user (be mindful of privacy, security)

Managing privileges for your user

You don't want to run everything as the root user, because this is insecure, but you also want to be able to run some things as root. How can we manage this without having to log in as root every time you want to run something?

/etc/sudoers

sudoers file - only edit through commands like `visudo`, because broken syntax will brick your system

Add your user/group to sudoers to give them the ability to escalate privileges through the "sudo" (superuser do) command

```
root@tornado:~# cat /etc/sudoers
#includedir /etc/sudoers.d

# We're using the /root principal in PAM so change the prompt accordingly
Defaults passprompt="%u/root's password: "

# Reset PATH, other environment variables, and umask
Defaults secure_path="/opt/share/utils/bin:/opt/share/utils/sbin:/usr/local/sbin:/usr/local/bin:/opt
/puppetlabs/bin:/usr/sbin:/usr/bin:/sbin:/bin"
Defaults env_reset
Defaults umask_override
Defaults !tty_tickets

# Hard code sudo access for root and system users
root ALL=(ALL) ALL
apt-dater ALL=NOPASSWD: /usr/bin/apt-get, /usr/bin/aptitude, /sbin/reboot, /usr/bin/dpkg
ocfbackups ALL=NOPASSWD: /usr/bin/rsync, /usr/bin/nice, /usr/bin/ionice, /usr/local/bin/rsync-no-van
ished
ocfdeploy ALL=NOPASSWD: /usr/local/sbin/puppet-trigger

# User specification
%ocfroot ALL=(ALL)  ALL

# Allow ocfstaff to run a few commands without password for easier scripting.
%ocfstaff ALL=(ALL) NOPASSWD: /usr/bin/virsh list, /usr/bin/virsh list --all

# Same with ocfroot
%ocfroot ALL=(ALL) NOPASSWD: /usr/local/sbin/puppet-trigger, /sbin/shutdown, /usr/bin/apt update, /u
sr/bin/apt-get update
root@tornado:~#
```

User/Group Takeaways

- Be very careful about creating users, and giving them permissions
- Limit file privileges as much as possible to minimize risk
- Know what the implications are when executing things as other users

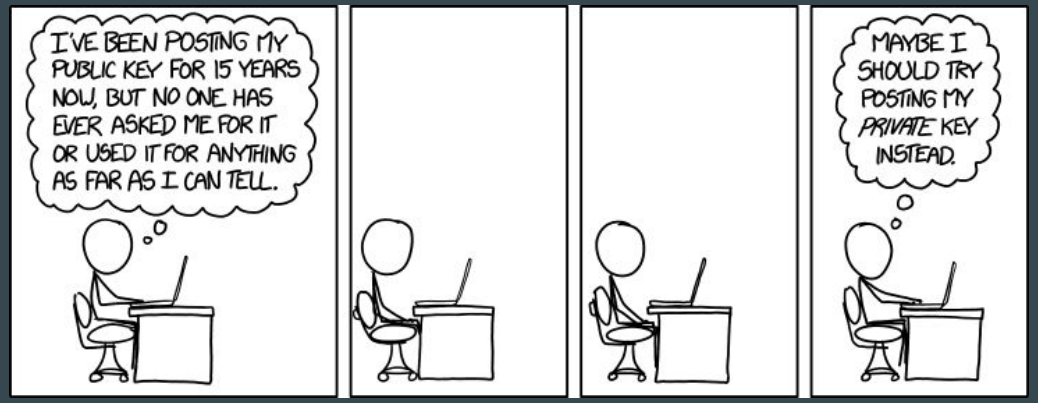
DON'T JUST PREPEND "SUDO" IF SOMETHING FAILS

(Advanced) - Extended Filesystem Stuff

- `lsattr` and `chattr` - beyond r/w/x for user/group/other, there are FS attributes you can edit
  - e.g. immutability - don't allow anyone to write to the file permanently, below --w--w--w-, or "undeleteability" - don't allow the file to actually be deleted, lots of other things that are FS dependent but accessible through `lsattr` and `chattr`

- `fsck` - the filesystem checker
  - For filesystems that support, for example, journaling, fsck will find orphaned files, missing inodes, etc. and help recover files that might have been lost on crashes/corruption. Equivalent to `chkdsk` on Windows.

SSH and Secure Access



Managing Secure Access

You've set up users and groups, now you need to manage remote access for those users. i.e. Secure Shell

Lots of ways to authenticate to SSH: password, SSH keys, GSSAPI (Kerberos) etc. By default, sshd will allow login for any user on the machine, including root

This could be dangerous - if a user has a weak password on an internet-facing system, especially if the root user is insecure, than the entire system can be compromised

Authentication and /etc/ssh/sshd_config

We edit /etc/ssh/sshd_config to fix some of these issues

- `PasswordAuthentication yes/no`
  - Allow passwords to be used to login (opens you up to brute-force password attempts)
- `PermitRootLogin yes/no/without-password`
  - Allow root user to login (allows brute-force to root, without-password = only keys allowed)
- `PubKeyAuthentication yes/no`
  - Allow using SSH keys for authentication (more secure than passwords)
- `GSSAPIAuthentication yes/no`
  - Generic Security Service API, = LDAP/Kerberos based authentication

Live Demo - make your SSH keys

1. Log into tsunami.ocf.berkeley.edu
2. If you don't already have an ~/.ssh/id_rsa, make your ssh key:
   a. ssh-keygen -t rsa -b 4096 (passphrase if you prefer, or not)
3. Copy it to your student VM
   a. ssh-copy-id <user>@<user>.decal.xcf.sh
4. Try to log in using your SSH key:
   a. ssh <user>@<user>.decal.xcf.sh
5. Add the host to your .ssh/config

```
Host <user>.decal.xcf.sh
    User <user>
    IdentityFile ~/.ssh/id_rsa
```

Reducing SSH Attack surface

1. Obviously, make sure that critical users have secure passwords
2. Use ssh keys as much as possible to avoid having passwords floating around
3. Bind ssh server to non-public interfaces, e.g. through a VPN
4. Inspect authentication logs to ensure no one is logging in that you don't expect:
    a. By default, /var/log/auth.log logs all authentication attempts
5. Use something like Fail2Ban on public networks - uses iptables or ufw to ban connections after 3-5 failed connection attempts
    a. After a while a significant part of the Chinese, Russian, and Eastern European IP ranges will probably be banned

Firewall Config

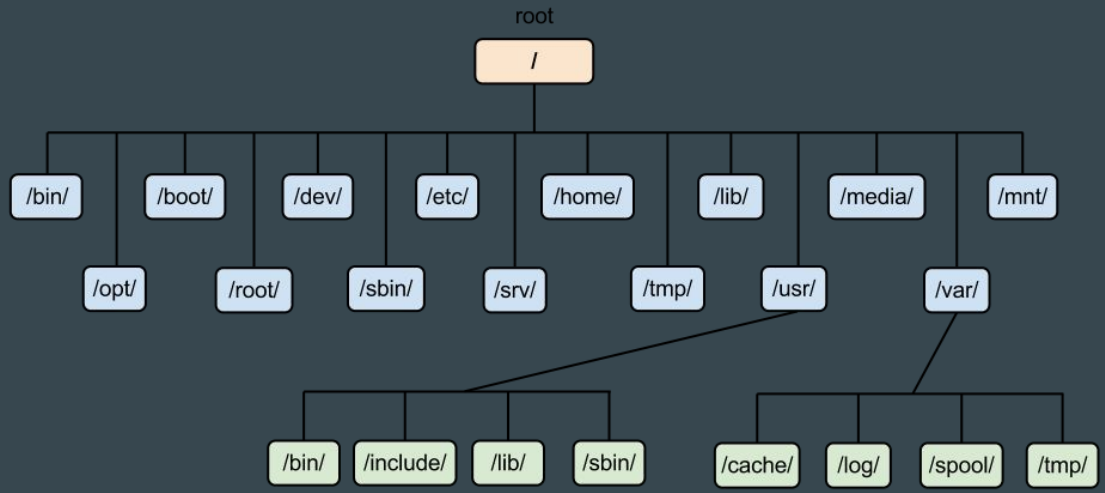
- Firewall restricts the connections that can be made to your machine.
- iptables: the standard Linux firewall. Tricky, complicated, potentially dangerous syntax.
- For personal servers, ufw (Uncomplicated FireWall) can make these rules for us.
  - `sudo ufw default deny`
    - Configure ufw to deny by default, highly recommend
  - `sudo ufw allow 80/tcp`
    - Allow connections on TCP port 80 (HTTP)
  - `sudo ufw allow SSH`
    - Allow connections on TCP port 22 (SSH)
  - `sudo ufw status`
    - Show list of all rules

Hi :)

...

Will be starting at 8:10

The Filesystem Hierarchy



Filesystem hierarchy

- **/boot** - files necessary for booting. Holds “vmlinuz,” the kernel executable!
- **/dev** - doesn’t hold actual “files,” instead abstracts away hardware **dev**ices.
- **/etc** - holds configuration files. Consider including this folder in your backups.
- **/home -** home directories for users on the system
- **/lost+found** - used by fsck (filesystem checker). Corrupted/recovered files show up here
- **/media** - automounted devices show up here
- **/opt** - holds files for packages that don’t conform to the hierarchy. Jupyter Notebook, Google Chrome, and some others install to this folder
- **/proc** - another “fake” filesystem, has information about all your processes

Filesystem hierarchy, continued

- **/root** - home folder for root user
- **/run, /tmp** - tmpfs. For files that don't need to persist across reboots.
- **/srv** - Intended to be used for programs that serve files (HTTP, FTP).
- **/sys** - Fake filesystem, only accessible by root. Gives information about system from the kernel.
- **/var** - "Things that change." Holds log files, database files.

Filesystem hierarchy, continued, continued

- /bin, /usr/bin, /sbin/ /usr/sbin/ /usr/local/bin, /usr/local/sbin, etc.
  - Binaries in your $PATH
- /usr/include - Header files for C programs.
- /lib, /usr/lib, /usr/local/lib - C libraries.
- /usr/share - Can be “shared” across machines.
- /usr/local/man - Manpages.

Extras: applications, dotfiles, and more setup

Basic Utilities

You might want to install some useful programs:

- tmux
  - Terminal multiplexer
- htop
  - htop gives you a graphical view of processes/systems/memory
- vim
  - The best editor
- yay, aurman, yaourt...
  - If on Arch, get an AUR helper to install community packages easily

Customize your dotfiles!

Dotfiles are user-specific configuration files that allow for a very deep level of customization. You should mess around with them to make your system feel like yours!

Some important ones:

- .bashrc: Runs every time you init a bash shell.
  - Set PATH, make aliases, change your color scheme...
- .vimrc: Make your vim awesome!
  - Install plugins, set custom keybinds, change behaviors
- .config/
  - Poke around in this folder to find application-specific configs

Resources

ArchWiki

Google

- Viewing system users https://www.digitalocean.com/community/tutorials/how-to-view-system-users-in-linux-on-ubuntu
- Users and groups https://wiki.archlinux.org/index.php/Users_and_groups
- UFW documentation https://help.ubuntu.com/community/UFW
- File permissions https://wiki.archlinux.org/index.php/File_permissions_and_attributes